Tutoriel d’utilisation de l’algorithme Isolation forest

Bienvenu dans ce tutoriel d’utilisation de l’algorithme Isolation forest. Ce tutoriel vous apprendra comment utiliser facilement cet algorithme pour analyser votre propre banque de données. Il vous fournira le code nécessaire mais aussi vous expliquera quels paramètres modifier si vous souhaitez modifier le code pour l’adapter à vos besoins.

1) Tutoriel

Pour utiliser ce tutoriel correctement il faut créer un fichier .py par exemple où on copie toute les parties bleues entre barres.

1 Formatage des données

Tout d'abord vous devez formater vos données afin qu'elles soient compatibles avec la suite de ce tutoriel.

Pour cela créez un tableau sur excel ou autre tableur. La première colonne est l'identifiant qui vous permettra d'identifier rapidement les anomalies qui vous seront indiquées, peut être le numéro de la ligne par exemple.

Ensuite chaque colonne dont on mettra le titre en ligne 1 contiendra les valeurs d'une caractéristique pour chaque élément de la base de données.

Finalement, enregistrez ce document sous le format **cvs séparé par des virgules** dans un dossier de votre choix.

2 Import des bibliothèques

Vous devez maintenant créer un fichier dans le MEME dossier que celui où vous avez mis votre dataset. Ce fichier est celui où nous allons écrire le programme une extension .py est donc conseillée.

Le fonctionnement de l’algorithme comme nous l’avons vu plus haut est assez complexe mais heureusement des bibliothèque assez simplifiée ont étés crées ce qui va nous simplifier grandement la tâche. Malheureusement ceci peut aussi nous désavantager car les seules fonctions qui sont données sont très opaque et ne laisse peut de place à la compréhension du code interne et à la modification de ce dernier. C’est pour cela que nous allons essayer de nous concentrer sur la compréhension des arguments que nous donnons aux fonctions que nous allons utiliser.

```
import numpy as np
```

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.ensemble import IsolationForest
```

Si vous voulez trouver les anomalies en prenant en compte une seule variable de votre base de donnez suivez la partie : "Recherche anomalie pour une seule variable". Si vous voulez connaitre les annomalies en prenant en comptes toutes les variables entrées dans votre documents csv suivez la partie : "Recherche anomalie pour toute les variable"

3 Recherche anomalie pour toute les variables¶

Dans cette section nous prenons en compte toute les variables entrées pour les éléments de notre dataset.

```
#Valeur que vous devez entrez

nom_fichier = ''


#code à ne pas modifier

df = pd.read_csv(nom_fichier)

df_modif = pd.read_csv(nom_fichier)

model=IsolationForest(n_estimators=50, max_samples='auto',
contamination=float(0.2),max_features=1.0)

model.fit(df)

df_modif['scores']=model.decision_function(df)

df_modif['anomaly']=model.predict(df)

anomaly=df_modif.loc[df_modif['anomaly']==-1]

anomaly_index=list(anomaly.index)

print(anomaly)
```

Veuillez entrer le nom du fichier entre les guillemets sur la première ligne. Cela correspond au nom du fichier créé en étape 1 contenant votre dataset.

3 BIS Recherche anomalie pour une seule variable

Si vous souhaitez savoir qu'elles sont les valeurs anormales pour une seule variables, par exemple ne prendre en compte que la variable de l'age dans une certaine population, il faut utiliser le code suivant.

```
#Valeur que vous devez entrez

nom_fichier = ''

nom_variable =''


# Reste du code à ne pas modifier

model=IsolationForest(n_estimators=50, max_samples='auto',
contamination=float(0.2),max_features=1.0)

model.fit(df[[nom_variable]])

model.predict(df[[nom_variable]])

df['scores']=model.decision_function(df[[nom_variable]])

df['anomaly']=model.predict(df[[nom_variable]])

anomaly=df.loc[df['anomaly']==-1]

anomaly_index=list(anomaly.index)

print(anomaly)
```

Veuillez entrer le nom du fichier entre les guillemets sur la premiere ligne. Cela correspond au nom du fichier créé en étape 1 contenant votre dataset. Ainsi que le titre de la colonne de la variable que vous voulez prendre en compte dans nom_variable.

4 Utiliser les résultats

Lancez le programme. Ce dernier va vous affichez les valeurs dites anormales. Vous pouvez ensuite agir sur ces dernières en utilisant la variable anomaly_index qui est un tableau qui contient tous les indices des anomalies détectées.

2) Explication plus précise du code

Voici une explication plus précise de chaque fonction utilisée dans le code donné dans le tuto.

1 Récuperation des données

Pour commencer, si nous avons bien suivi le tutoriel fournis nous avons déjà créée un fichier CSV qui rassemble nos donnés. La fonction suivante permet de stocker toute ces informations dans une variable en lisant le contenu de notre fichier.

```
df = pd.read_csv(nom_fichier)
```

2 Création du model

Notre but maintenant est de définir le modèle utilisé par l'algorithme Isolation Forest ensuite. Cela reviens à créer un objet contenant tous les paramètres donc la bibliothèque a besoin pour fonctionner.

Pour cela vous allez devoir choisir des paramètres suivants vos besoins :

1) n_estimators

Il s'agit ici de choisir le nombre d'estimateurs de base, c'est-à-dire le nombre d'arbres qui seront construits dans la forêt.

Plus le nombre d'arbre est élevé plus l'algorithme est fiable car les moyennes souffrirons moins des valeurs écartées mais ces calculs prennent du temps et son couteux c'est pour cela qu'un nombre trop élevé d'arbre serait du « gâchis » car la plupart du temps ils n'apportent rien car les anomalies étaient déjà isolées avec moins d'arbres.

On considère que 100 arbres sont suffisants pour quasiment toutes les applications courantes de recherche d'anomalies c'est d'ailleurs la valeur par défaut.

2)max_samples

Il s'agit ici de choisir l'échantillon max soit le nombre d'échantillons à tirer pour former chaque estimateur de base.

En effet lorsque l'on crée un arbre, surtout quand nous avons une grande banque de donnée nous n'utilisons pas forcément tous les éléments de la base de donnée pour chaque arbres. Un échantillon est créé pour n'utiliser que certaines données.

Vous pouvez changer cette valeur pour une valeur inferieur à 256 et aux nombres d'éléments dans votre base de donnée si vous avez vraiment une contraintes de nombre d'opération à effectuer mais celle si altèrera rapidement la qualité des résultats.

Vous pouvez mettre une valeur au dessus de 256 si votre base de donnée contient plus de 256 éléments et que vous n'avez pas peur d'une perte de rapidité et d'optimisation pour avoir un résultat plus précis.

Il est tout de même conseillé de laisser ce paramètre en paramétrage automatique celui-ci choisira la valeur min(256, n_samples) qui pour la plupart des application est nettement appropriée.

Le programme créera donc un nombre correspondant d'échantillons qui seront utilisés pour créer chaque arbres.

3) Contamination

Ce paramètre est le plus sensible et le plus délicat à choisir. Le taux de contamination est la proportion d'anomalies attendues dans la base de données.

Ce dernier est utilisé lors de l'ajustement pour définir le seuil sur les scores des échantillons. Il est essentiel car si nous affirmons une valeur pour ce paramètre le code nous donneras un nombre de valeur anormales correspondant. Par exemple si nous entrons contamination=float(0.2) le programme nous fournira 20% de valeurs aberrantes même si les scores d'anomalies en révèle plus ou moins.

La valeur par défaut est 'auto'. Si 'auto', la valeur seuil sera déterminée comme dans l'article original d'Isolation Forest.

Si nous connaissons cette valeur en revanche par exemple si nous savons d'avance que nous voulons enlever 10% des valeurs qui sont trop éloignées des autres nous pouvons l'indiquer tout en sachant qu'elle est comprise entre (0, 0.5]

4) max_features

Comme nous l'avons vu dans l'explication du principe de isolation forest, l'algorithme va créer des séparations parmi les données jusqu'à isoler les différents éléments.

Or il n'est pas forcements nécessaire de continuer de créer des séparations jusqu'à ce que tous les éléments de la base de données soit isolés. En effet comme nous avons vu les éléments les plus difficiles à isoler ne sont pas ceux qui sont considérés comme anormal. Le nombre de séparation créées peut donc être paramétré pour optimiser l'algorithme.

La valeur par défaut est 1.0, on peut prendre une valeur différente mais cette dernière doit être inferieur à 1.

5) bootstrap

Ce paramètre est un booléen, si on entre la valeur 'Vrai' cela indique que les tirages d'échantillonnage pour chaque arbre est effectué avec remise, si ce paramètre est 'Faux' cela indique cela indique que l'échantillonnage est fait sans remise.

Le réglage par défaut est False.

6) n_jobs

Ce paramètre permet de définir le nombre de tâche à exécuter en parallèle pour l'utilisation des fonction fit et predict que nous expliquons plus loin.

Il n'est pas utile de se soucier de ce paramètre à moins que nous ayons une dataset de très très grande taille.

Si sa valeur est 'none' une seule tâche est effectuée à la fois, si sa valeur est '-1' le programme utilisera alors tout le processeur disponible.

7) random_state

Ce paramètre contrôle le caractère pseudo-aléatoire de l'échantillonnage et du choix des splits, pour chaque étape de création d'arbre.

C'est un entier qui suivant sa valeur appellera les fonctions si dessous plusieurs fois afin de faire des moyennes des résultats pour accroire le caractère aléatoire des tirages faits.

8) warm_start

Ce paramètre est un booléen qui si il est paramétrer sur Vrai, réutilise la solution de l'appel précédent pour s'adapter et rajoute seulement des estimateur dans l'ensemble. Si le paramètre est Faux une toute nouvelle forêt est créée.

Ce paramètre peut être utile si vous utilisez une très grande base de donée et que vous pensez qu'une précision supplémentaire est nécessaire.

Les paramètres conseillés sont les suivants :

```
model=IsolationForest(n_estimators=50, max_samples='auto', contamination= »auto »
,max_features=1.0, bootstrap=False, n_jobs='None', random_state=None,
warm_start=False

)
```

4.3 Utilisation du model

Maintenant que nous avons défini le modèle avec lequel nous voulons travailler nous devons l'utiliser pour l'appliquer à nos données.

Pour cela nous allons utiliser la fonction fit qui va ajuster le modèle sur les données. En d'autres mots cette fonction crée tous les arbres de la forêt.

```
model.fit(df)
```

Nous utilisons ensuite la fonction decision_function qui va trouver le score d'anomalie de chaque éléments en étudiants pour chacun leurs positions dans tout les arbres créés précédemment. Elle va donc calculer le score d'anomalie moyen de X des classificateurs de base.

```
model.decision_function(df)
```

La fonction predict va ensuite attribuer une valeur a chaque éléments qui indique si cet élément est considéré comme normal (valeur 0) ou comme anomalie (-1) .

```
model.predict(df)
```

Pour plus de lisibilité nous plaçons ces deux scores dans notre tableau de données grâce aux fonctions suivantes :

```
df_modif['scores']=model.decision_function(df)
```

```
df_modif['anomaly']=model.predict(df)
```

Finalement nous cherchons un affichage correcte des résultats et pour cela nous isolons toute les lignes correspondants à des anomalies pour les mettre dans un tableau séparé. Nous en profitons pour isoler les index des anomalies dans une variable qui pourrait nous être utile plus tard.

```
anomaly=df_modif.loc[df_modif['anomaly']==-1]
```

```
anomaly_index=list(anomaly.index)
```

Et pour terminer nous affichons les lignes correspondants aux anomalies.

```
print(anomaly)
```

Pour visualiser les erreurs nous pouvons utiliser le code suivant.

```
# valeur à entrer
```

```
var1 = ''
```

```
var2 = ''
```

```
# Code à ne pas modifier
```

```
pred_scores = -1*model.score_samples(df)

plt.scatter(df[[var1]],df[[var2]], c=pred_scores, cmap='RdBu')

plt.colorbar(label='Simplified Anomaly Score')

show()
```

Veuillez choisir les deux noms des colonnes des variables que vous voulez choisir comme abscisse et ordonnées de votre graphique

Cet affichage permet de visualiser l'ensemble des données ainsi que leurs score d'anomalies associé. Cet affichages est très révélateur pour les base de données de dimensions 2 voire 3 mais pour les dimensions plus élevées cet affichage perd de son intérêt.